The World's Richest

Date: 2020-12-29

Chapter 1: Introduction

This project is to solve such a problem:

Given *N* people with their name, age and the net worth(or a short one, worth), and each query with M, Amin, Amax, print , the programme should output the 1^{st} ,2^{nd} ,...,M^{th} richest men whose ages lie in range [Amin,Amax].

When 2 men have the same worth, they should be non-decreasing ordered by their age. If they have the same age, then they should be non-decreasing alphabetical ordered by their name, in dictionary order. Every 2 men are different. In other word, there do NOT exist 2 men whose worth, age and name are all the same.

The formation of Input/Output is described below:

Input:	Output:
N K name[0] age[0] worth[0] name[1] age[1] worth[1] ... name[N-1] age[N-1] worth[N-1] M[1] A1[1] A2[1] M[2] A1[2] A2[2] ... M[K] A1[K] A2[K]	Case #1: (1^{st} richest man's) name age worth (age in [Amin[1],Amax[1]] (2^{nd} richest man's) name age worth (age in [Amin[1],Amax[1]] ... ($M[k]^{th}$ richest man's) name age worth (age in [Amin[1],Amax[1]] ...... Case #K: (1^{st} richest man's) name age worth (age in [Amin[K],Amax[K]] (2^{nd} richest man's) name age worth (age in [Amin[K],Amax[K]] ... ($M[k]^{th}$ richest man's) name age worth (age in [Amin[K],Amax[K]]

Moreover, if there are m[i], who is less than M[i], people whose ages are in [Amin[i],Amax[i]], just output m[i] people. Besides, if there is no person in such a age range, just output

Case #i: None

The input data restriction are also pointed out below:

$1 \le N \le 10^5, 1 \le K \le 10^3$, $|name| \le 8$, {char(name)}={a-Z, _}, {age}={1,2,...,200}, {worth}=$[-10^6,10^6]$(integer)

$M \le 100$, Amin,Amax in {age}, Amin≤Amax , Amin = min{A1,A2}, Amax = max{A1,A2}

Chapter 2: Algorithm Specification

2.0 Intro

Noticing that there are 3 attribute of a person : worth, age, name (ordered by their weight).

The idea of Table Sort hit me. Table Sort is used when the data structure is large. Well, 3 components, might be large.

Because there are M query restricted in age ranges, it is obviously that it would be better if we first do Bucket Sort. The result of Bucket Sort is 200 age buckets, each of which is a ordered list with the same age. Then for each query, we select buckets whose ages lies in given age range, combine them into a cluster, and sort them to find 1^{st} ,2^{nd} ,...,M^{th} richest men.

In most situations, M is less than the number of people in the cluster, so we cannot just sort the cluster. What we need is only the top M people. Well, Heap Sort naturally comes to give a helping hand, with its feature fitting the need.

Also, we could not slow down the speed of Bucket Sort. For the sorting in separate ages, I chose Quick Sort improved by Insertion Sort when the length of array is short(≤20).

Sorting Algorithms are listed : Table Sort, Bucket Sort, Quick Sort, Insertion Sort, Heap Sort.

2.1 Data Structure

2.1.1 person

The person is just combined by name, age and worth.

Conveniently, we use the pointers of structures instead of itself.

person

Component	Description
char name[NAMEMAX+1]	NAMEMAX = 8, 1 more char for '\0'
int age	age
int worth	net worth

Global variable *person* **Person** is a pointer pointing to person array

2.1.2 ageBucket

int * ageBucket[AGEMAX+1];*//[0,AGEMAX], 0 unused*

For each age, **ageBucket[age]** is *int* * , pointing to an int array whose content is the Order Table.

That is, **ageBucket[age][0]** is the richest (in the scope of men aged **age**) man's index of the array *person* **Person.**

2.1.3 heap

*typedef int * heapNode;//pointer pointing to the index of the array person* **Person.**

heap

Component	Description
int size	the size of heap, count of nodes currently existing, initially to be the capacity of the heap
heapNode arr	the storage space for nodes, aka *int* * (index's pointer)

2.2 Bucket Table Sort

2.2.1 create Bucket Table

Bucket : Count the number of people with distinct ages, allocate proper memory for each age Bucket.

Table : The content of each Bucket is man's index of the array *person* **Person**, and each Bucket ends with -1.

-1 , who acts like '\0' in the char array(string),is a boundary telling that it's the end. In other words, the people aged i are all visited when -1 appears.

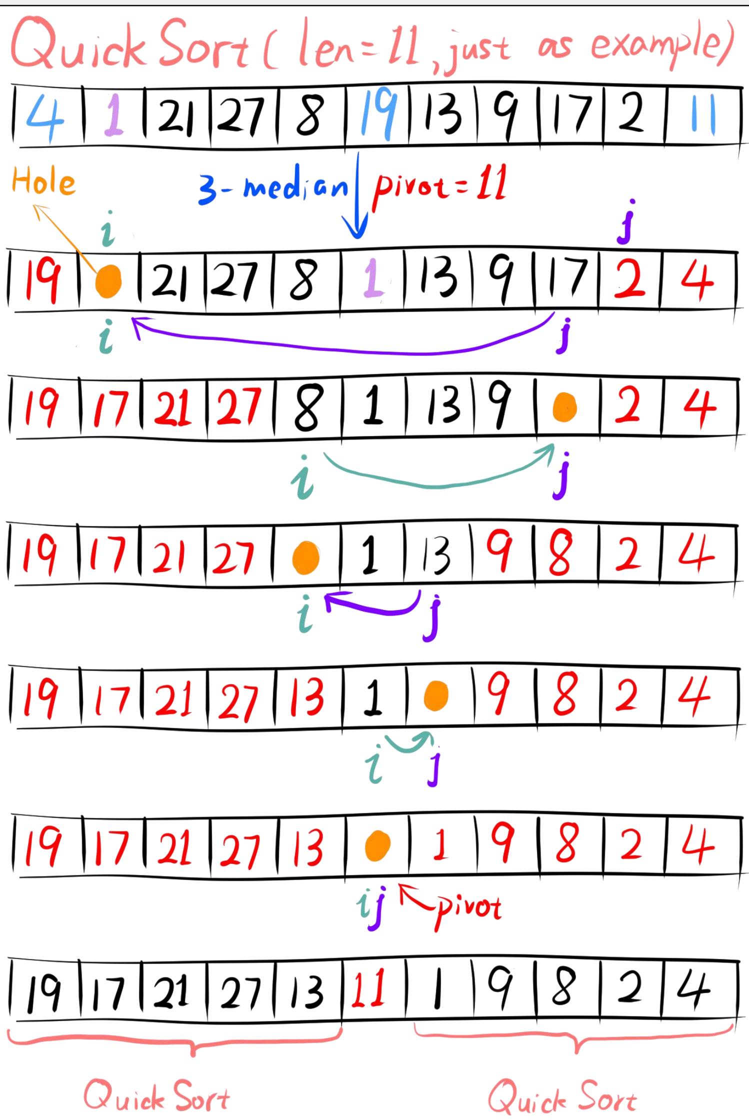
2.2.2 divide into Buckets

Put man's index of the array *person* **Person** into their age Bucket.

2.2.3 Sort in each Bucket

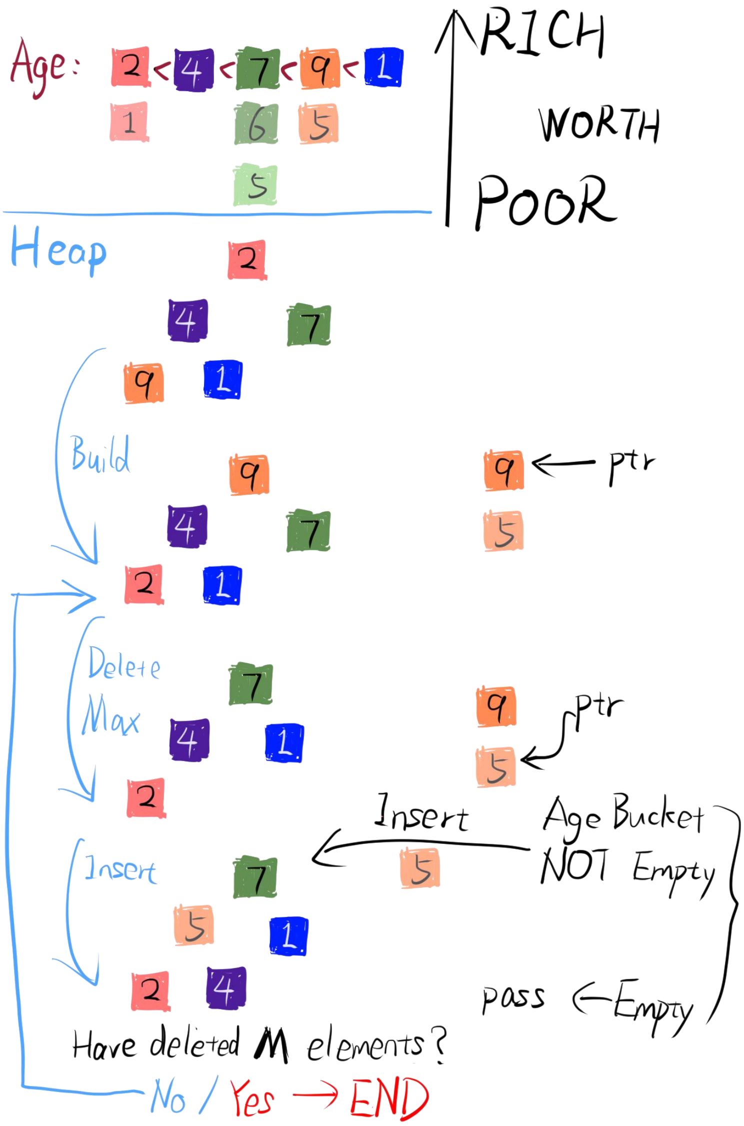
```
void Sort(int age)
{
        quickSort(age,0,ageCnt[age]-1);
}
```

```
void quickSort(int age,int l,int r)
{
        int len = l - r;
        if(len<=20){InsertionSort(age);}
        else//quickSort, 3-median
        {
            int median = (l+r)/2;
            // See the following diagram
        }
}
```



2.3 Cluster Heap Sort

Cluster : compose of several age Buckets : age lies in [Amin,Amax], shown in the left-top part of the following diagram. And the idea of Algorithm is drawn in the diagram. As for the implementation of Heap Operation, they have been discussed in the lecture, which should be omitted here.



Chapter 3: Testing Results

Description of a Graph	Purpose of this case	Expected Result	Current Status
12 4 Zoe_Bill 35 2333 Bob_Volk 24 5888 Anny_Cin 95 999999 Williams 30 -22 Cindy 76 76000 Alice 18 88888	Sample by the Problem	Case #1: Alice 18 88888 Billy 24 5888 Bob_Volk 24 5888 Dobby 24 5888 Case #2: Joe_Mike 32 3222	Pass

Joe_Mike 32 3222 Michael 5 300000 Rosemary 40 5888 Dobby 24 5888 Billy 24 5888 Nobody 5 0 4 15 45 4 30 35 4 5 95 1 45 50		Zoe_Bill 35 2333 Williams 30 -22 Case #3: Anny_Cin 95 999999 Michael 5 300000 Alice 18 88888 Cindy 76 76000 Case #4: None	
1 1 Nobody 5 0 2 4 6	Boundary	Case #1: Nobody 5 0	Pass
50 3 jiiuo 25 1678 kjpsc 25 -6227 wwkxd 25 -3127 cqllh 25 1029 oxqub 25 -1021 ikitg 25 1297 mtwri 25 -3731 xksuk 25 -3787 fboxx 25 -7001 waswn 25 -8628	Random Case with the same age, Query 1, A1=A2 Query 2, A1>A2 Query 3, A1<A2, No person's age in [A1,A2]	Case #1: vbpis 25 9538 baklc 25 9311 hwlab 25 8557 rxifv 25 7906 utpnt 25 5841 Case #2: vbpis 25 9538 baklc 25 9311 hwlab 25 8557 rxifv 25 7906	Pass

vhyqr 25 −8906		utpnt 25 5841	
aetpc 25 −9094		vfzxh 25 5825	
ehtlv 25 −1348		huuns 25 5362	
cehzg 25 −9516		rwspl 25 5296	
rxifv 25 7906		ocuwp 25 5293	
ocuwp 25 5293		ekxga 25 4959	
jplwh 25 364		dvgha 25 4861	
rpetg 25 −227		whzia 25 3931	
djpbf 25 −8283		jvbug 25 3738	
qqflj 25 −1154		anepf 25 3090	
whzia 25 3931		jiiuo 25 1678	
utpnt 25 5841		epmts 25 1626	
ffqfh 25 −1222		ikitg 25 1297	
vnman 25 −4983		cqllh 25 1029	
nbjnq 25 −8752		bcmov 25 969	
ogbyi 25 −2211		nozvi 25 603	
rflqo 25 −2170		pjjmo 25 377	
jvbug 25 3738		jplwh 25 364	
vbgvq 25 −1814		haqjf 25 50	
tepxc 25 −6450		rpetg 25 −227	
bcmov 25 969		oxqub 25 −1021	
rwspl 25 5296		qqflj 25 −1154	
haqjf 25 50		ffqfh 25 −1222	
epmts 25 1626		ehtlv 25 −1348	

dvgha 25 4861 jiavg 25 −6276 epuya 25 −7128 vfzxh 25 5825 huuns 25 5362 nozvi 25 603 ykfpj 25 −1433 baklc 25 9311 xrmlk 25 −9004 vbpis 25 9538 qispg 25 −8830 pjjmo 25 377 anepf 25 3090 hwlab 25 8557 dxvzh 25 −5317 ekxga 25 4959 5 25 25 50 26 12 2 23 24		ykfpj 25 −1433 vbgvq 25 −1814 rflqo 25 −2170 ogbyi 25 −2211 wwkxd 25 −3127 mtwri 25 −3731 xksuk 25 −3787 vnman 25 −4983 dxvzh 25 −5317 kjpsc 25 −6227 jiavg 25 −6276 tepxc 25 −6450 fboxx 25 −7001 epuya 25 −7128 djpbf 25 −8283 waswn 25 −8628 nbjnq 25 −8752 qispg 25 −8830 vhyqr 25 −8906 xrmlk 25 −9004 aetpc 25 −9094 cehzg 25 −9516 Case #3: None	

See ../code/input.txt	Random Test Case Generated by RTCG.py	See ../code/output.txt	pass
EXTENSION: N,K is positive integer according to the description, but what if one of them is 0?			
0 1 5 12 89	Extreme Boundary without Person	Case #1: None	Pass
3 0 dad 35 10000 mom 35 10000 me 12 20	Extreme Boundary without Query	/*NO QUERY NO OUTPUT*/	Pass
0 0	Extreme Boundary with neither Person nor Query	/*NO QUERY NO OUTPUT*/	Pass

Chapter 4: Analysis and Comments

4.1 Space Complexity

4.1.1 People & Bucket Table Sort

N people : each person has 3 components, S=O(N)

AGEMAX different ages : each age has 1 element, ageCnt, S = O(AGEMAX)

Bucket Table : each person has one index, S = O(N)

Quick Sort using iteration : the number of stack levels < **log** the length of each age bucket< **log** N, S= O(**log** N)

4.1.1 Cluster Heap Sort

Heap array :

Denote count of ages with at least one person where age lies in [Amin,Amax] by A

Heap array are at most A heap node long. S = O(A)

4.2 Time Complexity

4.2.1 Bucket Table Sort

create Bucket Table and divide into Buckets : iterate *person* **People** twice, count and copy, T = O(N)

Quick Sort with Insertion Sort when length is short:

Quick Sort(3-median version) : In lecture, we have proved $T_{worst}=O(N^2)$, $T_{best} = T_{average} = O(N \log N)$

Insertion Sort make it quicker to sort the array.

Because N people are divided into AGEMAX=AM buckets,

in average, $T_{average} = AM\ O[(N/AM) \log (N/AM)] = O[N \log (N/AM)]$

Due to AM=200<<N, $T_{average,total} = O(N)+O[N \log (N/AM)] = O(N)+O(N \log N) = O(N \log N)$

4.2.2 Cluster Heap Sort

The Heap have A nodes, each deleteMax or insertion has O(log A) time complexity.

At worst case, for each person we get ,we all need to delete Max and insert the next one on the ageBucket. We need top M richest people, therefore T = M*2O(log A) = O(M log A)

We have K queries, and each query's M differs.

$T_{total} = \Sigma(i:1,...,K)\ O(M[i] \log A[i])$

Usually, $M \le 200 \ll K$, $A \le AM \le 200 \ll K$, therefore $T_{total} = O(K)$

Appendix: Source Code (in C)

see ../code/*.c

Declaration

I hereby declare that all the work done

in this project titled "The World's Richest"

is of my independent effort.